# Chapter 1
# Dynamic Load Balancing for Finite Element Calculations on Parallel Computers*

Eddy Pramono† Horst D. Simon ‡ Andrew Sohn§

**Abstract**

Computational requirements of full scale computational fluid dynamics change as computation progresses on a parallel machine. The change in computational intensity causes workload imbalance of processors, which in turn requires a large amount of data movement at runtime. If parallel CFD is to be successful on a parallel or massively parallel machine, balancing of the runtime load is indispensable. Here a frame work is presented for dynamic load balancing for CFD applications, called Jove. One processor is designated as a decision maker Jove while others are assigned to computational fluid dynamics. Processors running CFD send flags to Jove in a predetermined number of iterations to initiate load balancing. Jove starts working on load balancing while other processors continue working with the current data and load distribution. Jove goes through several steps to decide if the new data should be taken, including preliminary evaluate, partition, processor reassignment, cost evaluation, and decision. Jove running on a single SP2 node has been completely implemented. Preliminary experimental results show that the Jove approach to dynamic load balancing can be effective for full scale grid partitioning on the target machine SP2.

## 1 Introduction

The computation of the airflow over a complex aircraft in three dimensions is a challenging task. It is particularly challenging, when unstructured grids and finite element or finite volume methods are going to used [2]. The computational requirements for such a problem can only be satisfied by massively parallel machines [4, 8]. During a typical adaptive, unsteady CFD calculation, these type of unstructured meshes will have to be locally refined, and derefined, depending on the characteristic of the calculation, e.g. in order to capture a moving shock. The computational intensity over regions changes in time and in some regions near to body it is typically much higher than far away from the body. A parallel implementation of these finite element or finite volume methods on parallel machines typically entails two steps [9]: First, as a preprocessing step the computational mesh (or grid) is partitioned into smaller meshes. Second, the partitioned submeshes are then mapped to processors based on a mapping strategy. While this static partitioning and mapping approach would be appropriate for steady CFD calculations

*This work was supported through NASA Contract NAS 2-12961.

†Weidlinger and Associates, 4410 El Camino Real, Suite 110, Los Altos, CA 94022, pramono@wai.com

‡Computer Sciences Corporation, Mail Stop T27A-1, NASA Ames Research Center, Moffett Field, CA 94035, simon@nas.nasa.gov

§Dept. of Computer and Information Science, New Jersey Institute of Technology, Newark, New Jersey 07102-1982, sohn@cis.njit.edu. This work was supported in part by NASA/University Joint Venture in Research Program. This work was performed while the author was at NASA Ames Research Center.

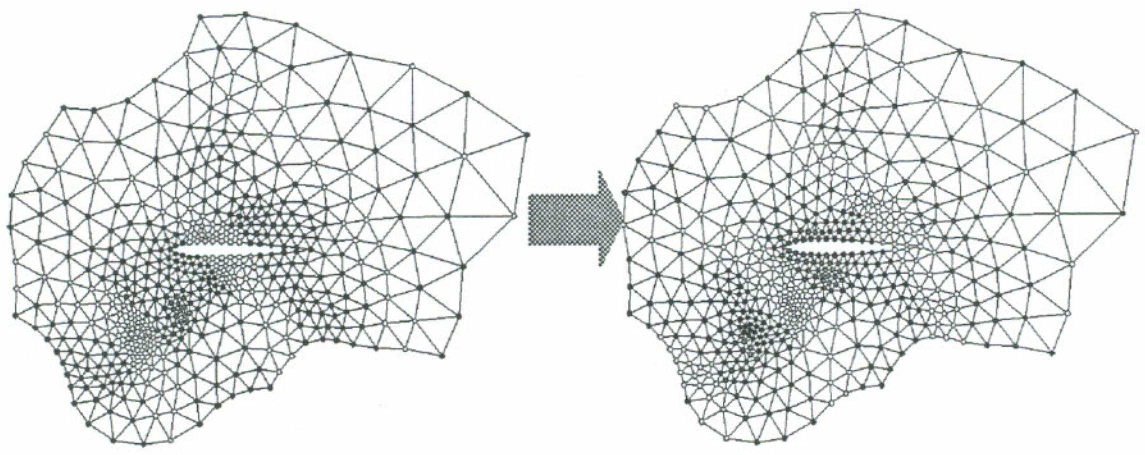
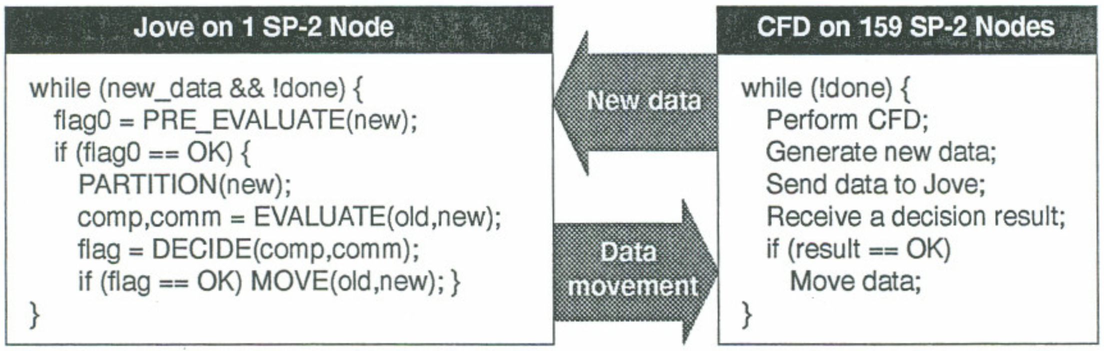


FIG. 1. *Two successive snap shots of an airfoil grid with changing weights.*

which do not change computational intensity during the iteration, this static approach is simply inefficient for unsteady, adaptive calculations. As the computation progresses on all processors simultaneously, the CFD problem running on a multiprocessor changes its computational behavior. Some processors will have a lot of work to do while some little, or often it is possible that some have no work to do. The work processors perform is not balanced and the initial partition is no longer efficient due to changes in the computation. It is therefore indispensable that the amount of work each processor performs be balanced at runtime to increase the processor utilization and performance [3]. Balancing runtime computational load, however, is often very difficult due to many practical issues such as measurement of computational load, decision of workload distribution, runtime data movement, minimization of communication, etc. It is precisely the purpose of this report to establish a framework of runtime load balancing for CFD-related applications. We investigate how the amount of work each processor executes can be balanced at runtime. The dynamic load balancer Jove addresses load balancing along computational intensity for workload distribution, and communication intensity for data distribution. A processor designated as Jove monitors computational load while all other processors execute a CFD application. Changes in computational behavior are monitored by Jove in every predetermined number of iterations. The Jove processor then evaluates if moving data from processors to processors will be beneficial. If data movement is costly compared to computational gain, no data movement will take place and the CFD continues without interruption. Should Jove decide that the new data set has intolerable change in computational intensity, it will instruct processors to move data according to the Jove's determination. In this short report we will give only an overview of Jove. The details will be found in [5].

## 2 Dynamic Load Balancing

A typical structural analysis or computational fluid dynamics problem is represented as a grid of vertices $V$ and edges $E$. A vertex representing a finite element has computational



```
while (new_data && !done) {
  flag0 = PRE_EVALUATE(new);
  if (flag0 == OK) {
    PARTITION(new);
    comp,comm = EVALUATE(old,new);
    flag = DECIDE(comp,comm);
    if (flag == OK) MOVE(old,new); }
}
```

```
while (!done) {
  Perform CFD;
  Generate new data;
  Send data to Jove;
  Receive a decision result;
  if (result == OK)
    Move data;
}
```

FIG. 2. *Jove, the load balancer.*

intensity associated with it, often called weight, or simply $W$. Figure 1 shows a very small scale airfoil grid $G$ consisting of 460 vertices and 1303 edges. The darkness of vertices represent computational intensity (weight). An edge connecting two vertices represents the existence of a computational relation between the two finite volumes (see [9] for more details). Note that although the grid in Figure 1 is a real finite volume CFD grid, we have used artificially created weights, demonstrating here just the conceptual framework for our load balancing scheme. A parallel implementation of CFD on multiprocessors requires the grid of vertices and edges to be divided into smaller grids, each of which is assigned to a processor. The degree of connectivity and computational weights of vertices determine how adjacent vertices should be grouped to form smaller grids (or meshes). The division of vertices of variable computational weights must be carefully done such that the resulting subgroups must have an equal or nearly equal amount of computational intensity. We make here the very important assumption, that the change in computational requirements in the CFD calculation can be adequately modeled by the weights alone. This is certainly true, when one starts out with a coarse grid, which is then subsequently refined and derefined. If, for example, a triangle is refined into four subtriangles in a two-dimnesional calculation, then this will be modeled by multiplying the weight of the original triangle by four. This model of using changing weights, however, cannot accomodate global changes in the grid connectivity, e.g., contact in crash codes, where previously disconnected portions of the mesh start interacting. What makes parallel CFD difficult is change in weight (computational intensity). As computation progresses, the weight associated with vertices changes in time due to different computational requirements. Figure 1 shows two grids, $G_i$ and $G_{i+1}$, separated by an iteration. A grid $G_i$ with $W_i$ changes to a new grid $G_{i+1}$ with $W_{i+1}$ after an iteration. The smaller grids partitioned and assigned to processors at the beginning of computation are effective only for a short period of time. After a few iterations of CFD computation, the majority of vertices will have their weights changed and the computational intensity of each processor will be different, resulting in load imbalance. Note that the small filled circles at the lower left corner of $G_i$ have moved toward the upper right corner of $G_{i+1}$ after an iteration.

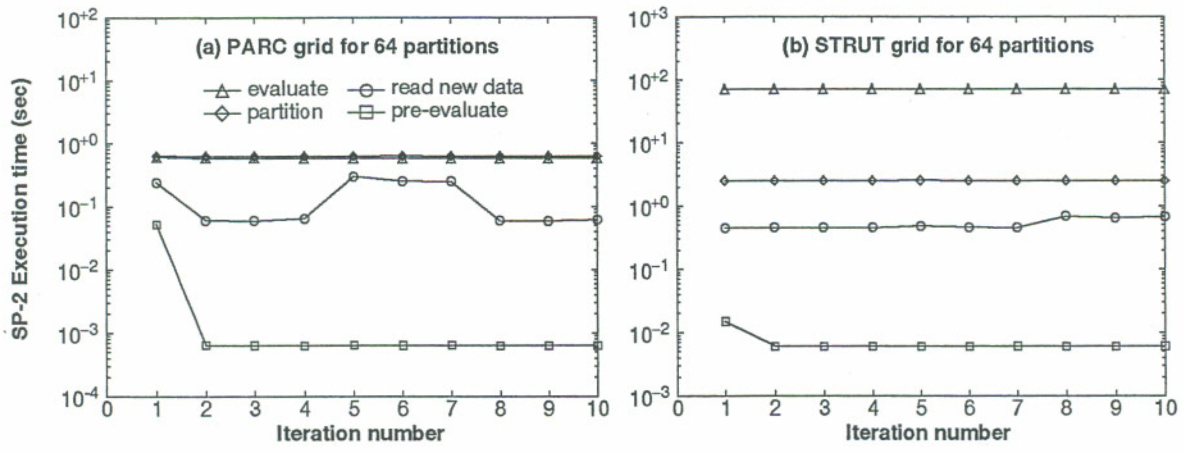


FIG. 3. *Execution time (sec) on an SP-2 processor.*

The two snap shots are made to the simplest to demonstrate the change in computational intensity. Only four different types of weight are illustrated, including hollow circle for the lowest intensity, filled circle for the highest intensity, and two different shaded circles for intermediate intensity. As the two snapshots shown above suggest, it is extremely important to dynamically repartition the grid according to new computational weights. It is however not straightforward to repartition the grid at runtime. There are many practical difficulties. Partitioning new weights into new smaller grids must be quick such that newly partitioned grid can be evaluated in time while CFD computation continues working on the next iteration. Evaluation and decision steps must be able take communication loss and computational gain into account in deciding whether the newly partitioned grids will give the best possible balanced load to all processors. The approach presented here is centered around the two premises: First, the partitioning of newly generated grid is quick compared to CFD computation time. Second, the decision of accepting newly partitioned grids is based entirely on the evaluation of computational gain and communication loss. Figure 2 gives an overview of our approach to dynamic load balancing Jove. The entire system consists of two modules: the decision maker Jove running on an SP-2 processor and the CFD module running on 159 SP-2 processors. (We explain Jove here in the context of the 160 node IBM SP-2 at NASA Ames, however the discussion equally applies to any other MIMD message passing computer). Jove executes the while loop shown on the left hand side of Figure 2 while the CFD application executes the one on the right hand side. Details of CFD application are beyond the scope of this report and will not be explained further. The first step of Jove is pre-evaluate which decides if the new data generated by CFD should be taken for further consideration. The objective behind preliminary-evaluate is to find if the newly generated weights are bad enough to be considered for repartitioning. If the preliminary evaluation deter mines that the new weights are indeed bad, it will recommend the new weights for repartition. The basic criterion for the decision of good/bad partition is to use the imbalance factor derived from the projection of new data to old partition.

The partition step divides a grid into sets. This step uses a modification of the successful Recursive Spectral Bisection (RSB) algorithm [1, 7], which uses an inertial approach to rapidly update a partition from one grid to the next one. This new algorithm called Dynamic Spectral Bisection will be explained in detail in [6].

Given now that vertices are partitioned into $n$ sets, the evaluation step proceeds, which consists of assignment and cost computation. The assignment step includes processor assignment and set assignment. Processor assignment ensures that each processor receives the same number of partitions. Set assignment ensures that each set has a unique label assigned to it. No two sets will, therefore, have the same set number. Cost computation finds computational gain and communication loss for the newly generated weights. These two values are computed based on the processor assignment and set assignment. The decision step simply decides if the new weights should be taken. If the computational gain is larger than communication loss, the decision will be to take the newly generated weights. Otherwise, the newly generated weights are discarded and Jove waits for the next weights. Should the decision step accept the new weights, data movements will take place according to the set and processor assignments.

## 3 Preliminary Results

The above approach has been implemented on an SP-2 multiprocessor installed at NASA Ames Research Center. Jove, written in C with approximately 3000 lines of C code, is running on a SP2 processor. CFD application programs are currently being completed and are expected to be connected with Jove in the near future.

Two model unstructured grid problems were used for experiments. The first grid, PARC, has 1240 vertices and 3355 edges while the second grid, STRUT, has 14504 vertices and 57387 edges. All the snap shots (in termediate grids) are prepared as files which are fed to Jove for decision. Figure 3 illustrates a typical behavior of Jove on an SP-2 processor for the two grids. The x-axis plots iteration numbers while the y-axis shows execution time for each iteration. Note that y-axis is plotted to logarithmic scale and that the two plots have different scale due to much different computation time.

Figure 3 shows that partition and evaluate time are large and dominate the whole process. The preevaluate step is essentially negligible as it is a fraction of evaluate time. The read time fluctuates due to accessing files on disk (file access will be replaced by runtime data when the CFD applications are complete). When the number of vertices increases, evaluate time starts dominating the whole process. The left diagram in Figure 3 shows that evaluate time entirely dominates the whole process. We stated earlier that we wish to quickly partition at runtime. The objective is certainly achieved as Figure 3 indicates. However, the results also indicate that evaluate time is too large compared to partition time. This clearly indicates that finding an optimal processor assignment will be difficult because the whole process will depend much on the evaluation step.

The two plots show that the execution time of individual steps do not drastically change over iterations. In fact, they remain almost constant, which is an encouraging news since an actual CFD would take hundreds or thousands of iterations before termination. Two plots also reflect the relation between the number of vertices and execution time. We find that the execution time for STRUT is an order of magnitude greater than that for PARC. STRUT has indeed 10 times more vertices than PARC. The relationship between number of vertices and execution time is an important factor in performing CFD on a multiprocessor since it will help identify the behavior of Jove, which in turn will affect the decision on data movement. Note that the above plots do not take data movement into consideration as this report focuses mainly on establishing a framework for dynamic load balancing.

We are currently interacting with application scientists, translating an MPL (message passing library) version of CFD applications to an MPI (message passing interface) version for the 160-processor SP-2. This work is expected to be completed in the near future. Once full scale CFD is running on the SP-2, we will be able to precisely measure the performance of a dynamic load balancer Jove. The approach described in the report will be a basis for balancing load for future parallel implementations of CFD at the NASA Ames Research Center. We anticipate that this framework will not only be applicable for CFD applications but more importantly for various grand challenge problems on massively parallel machines.

## References

[1] S. T. Barnard and H. D. Simon, *A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems*, Concurreny: Practice and Experience, 6 (1994), pp. 101 – 117.

[2] T. J. Barth and H. E. Deconinck, *Unstructured Grid Methods for Advection Dominated Flows*, AGARD, 7 Rue Ancelle, 92200 Neuilly-sur-Seine, France, 1992.

[3] K. Devine, J. Flaherty, S. Wheat, and A. Maccabe, *A massively parallel adaptive finite element method with dynamic load balancing*, in Proceedings of Supercomputing '93, Portland, Oregon, Los Alamitos, California, 1993, IEEE Computer Society Press, pp. 2 – 11.

[4] P. Mehrota, J. Saltz, and R. e. Voigt, *Unstructured Scientific Computation on Scalable Multiprocessors*, MIT Press, Cambridge, Mass., 1992.

[5] H. Simon and A. Sohn, *Dynamic load balancing for grid partitioning on a sp-2 multiprocessor: A framework*, Tech. Rep. RNR-94-xxx, NASA Ames Research Center, Moffett Field, CA 94035, October 1994. (in preparation).

[6] ——, *Dynamic spectral partitioning*, Tech. Rep. RNR-94-xxx, NASA Ames Research Center, Moffett Field, CA 94035, October 1994. (in preparation).

[7] H. D. Simon, *Partitioning of unstructured problems for parallel processsing*, Computing Systems in Engineering, 2 (1991), pp. 135 – 148.

[8] H. D. e. Simon, *Parallel CFD - Implementations and Results Using Parallel Computers*, MIT Press, Cambridge, Mass., 1992.

[9] V. Venkatakrishnan, H. Simon, and T. Barth, *A MIMD implementation of a parallel euler solver for unstructured grids*, The Journal of Supercomputing, 6 (1992), pp. 117 – 127.